CODE GENERATION

- Translation of software in high-level code (like C) to machine instructions

- Based on (second part of) following paper:

WHY DIFFICULT?

- Code generated for C compilers for PDSPs (programmable digital signal processors) is several factors slower than assembly code.
- Reason: PDSPs have a data path that is less regular than conventional processors (more parallelism, special-purpose registers).

TEXAS INSTRUMENTS TMS320C25

- Features:
  - Address generation unit (AGU)
  - Temporary register (TR)
  - Product register (PR)
  - Accumulator (ACCU)
  - Multiply-accumulate instruction

TI TMS320C25 dates from 1987-1990
TRADITIONAL COMPILATION PROCESS

- Source program
- Source code analysis
- Intermediate representation (IR)
- Machine-independent IR optimizations
- Optimized intermediate representation
- Code generation
- Assembly program

SOURCE-CODE ANALYSIS

- Lexical analysis:
  - Group characters into tokens.
  - Can be automated by programs like `lex (flex)`.
- Syntax analysis:
  - Apply grammar rules and identify constructions.
  - Results in syntax tree, a data structure explicitly showing expressions, statements (conditionals, loops).
  - Can be automated by programs like `yacc (bison)`.
- Semantical analysis:
  - Identify scopes of variables, etc.

TRADITIONAL MACHINE-INDEPENDENT IR OPTIMIZATIONS

- Constant folding:
  - Simplify constant expressions.
- Common-subexpression (CSE) elimination:
  - Calculate CSEs only once.
- Loop-invariant code motion:
  - Move code outside loop, when code does not depend on loop state
- Etc.

TRADITIONAL MACHINE-DEPENDENT IR OPTIMIZATIONS

- Code selection:
  - Select a minimum set of instructions to implement IR primitive.
- Register allocation:
  - Select registers for storage of intermediate results.
- Instruction scheduling:
  - Order the selected machine instructions.
  - Avoid spill code, moving values from registers to memory and back due to insufficient number of registers.
PROBLEMS OF TRADITIONAL APPROACH

• Irregular register location:
  – Better combine register allocation with code selection.
• Instruction-level parallelism (ILP):
  – Many instructions can be scheduled simultaneously.
  – Opportunities for code compaction.

PROPOSAL FOR CODE GENERATION

• Sequential code generation:
  – First ignore parallelism.
• Memory-access optimization:
  – Code for AGU.
  – Partition variables across multiple memories, accessible in parallel.
• Code compaction:
  – Try to merge sequential code into instructions.

SEQUENTIAL CODE GENERATION

• Represent computation to be compiled by data-flow trees (DFTs) or data-flow graphs (DFGs)
• Represent instructions by small DFTs: instruction patterns
• Try to optimally cover the computation graph by instruction patterns.
• Pay attention to registers (represent individual registers explicitly in register patterns).

EXAMPLE OF DFG COVERING

```c
int a,b,c,d,x,y,z;
Void f()
{
    x = a - b;
    y = a - b + c * d;
    z = c * d;
}
```
Use of MAC does not help as result of multiplication is also needed.

DFG-TO-DFT CONVERSION

- Reduces covering complexity at the expense of optimality

REGISTER-SPECIFIC PATTERNS

accu: PLUS(accu, mem)
accu: PLUS(accu, pr)

MEMORY BANK PARTITIONING

- Many DSP families not only have separate data and program memories (Harvard architecture), but two data memories often called X and Y.
- Assigning data to either X or Y is an optimization problem:
  - Data to be accessed at the same time should reside in different memories.
FREESCALE/MOTOROLA/NXP DSP56600

Example of architecture supporting multiple data buses

56000 series start in 1986, still in use

ADDRESS-GENERATION UNIT (AGU)

- Instructions:
  - AR load
  - MR load
  - AR modify
  - Auto-increment
  - Auto-modify
- Zero-cost:
  - Means address computation parallel to other instruction

MEMORY-ACCESS OPTIMIZATION

- It is a good idea to maximize zero-cost operations by a clever storage of values in memory:
  - Find a Hamiltonian path in access graph.
- Example on next slide has one AR available.
**MAXIMUM HAMILTONIAN PATH**

- Construct access graph:
  - Weighted graph
  - Weight is number of accesses neighboring in time

- Process of merging instructions to exploit the parallelism present in the PDSP.
- Variant of “resource-constrained scheduling”.
- One needs to take into account:
  - Data dependencies: no read of variable before write.
  - Anti-dependencies: no overwrite before last read.
  - Output dependencies: no simultaneous write to same location.
  - Incompatibility constraints: hardware limitations, instruction-format restrictions.

```c
int ar, ai, br, bi, cr, ci;
cr = ar*br - ai*bi;
ci = ar*bi + ai*br;
```

**INCLUDING ADDRESS GENERATION**

```
+-----------------+-----------------
|   0            |    ci          |
|   1            |     br         |
|   2            |     ai         |
|   3            |     bi         |
|   4            |     cr         |
|   5            |     ar         |
+-----------------+-----------------`

LARK 5  // load AR with &ar
LT *   // TR = ar
SBRK 4  // AR = 4 (&br)
MPY +   // PR = TR * br, AR++ (&ai)
LTP +   // TR = ai, ACCU = PR, AR++ (&ai)
MPS +   // PR = TR * ai, AR++ (&ai)
SPAC   // ACCU = ACCU - PR
STSB    // TR = ar
SBRK 2  // AR = 2
MPY -   // PR = TR * ai, AR-- (&ai)
LTP -   // TR = ai, ACCU = PR, AR-- (&ai)
MPS -   // PR = TR * bi, AR-- (&ci)
APAC   // ACCU = ACCU + PR
STSB    // TR = ar
```
RETARGETABLE CODE GENERATION

- Processor model is external to compiler.
- Low effort to adapt to new processor architectures.
- Helps to speed up design-space exploration:
  - Applications can be compiled for many processor variants;
  - Performance of each variant (area, speed, power) can be evaluated relatively easily.

The University of Twente has licenses for:

**Synopsys ASIP Designer**

(the new name of the *Target* tool suite as presented in [Goo05]).

ARCHITECTURAL SCOPE FOR PROCESSOR DESIGN

- Data types
- Arithmetic functions
- Memory organization (von Neumann vs. Harvard)
- Instruction format (encoded vs. orthogonal)
- Registers (homogeneous vs. heterogeneous)
- Instruction pipeline
- Control flow